Exhibit D

	Class Overview `Policy` is the common super type of classes which represent a system security policy. The `Policy` specifies which permissions apply to which code sources. Android Developer Tools available at http://developer.android.com/reference/java/security/AccessController.html: **Class Overview** `AccessController` provides static methods to perform access control checks and privileged operations. Android Developer Tools available at http://developer.android.com/reference/java/security/AccessControlContext.html: **public AccessControlContext (AccessControlContext acc, DomainCombiner combiner)** *** If a `SecurityManager` is installed, code calling this constructor needs the `SecurityPermission` `createAccessControlContext` to be granted, otherwise a `SecurityException` will be thrown. *See also* PolicyEntry.java: In Froyo: /** * Constructor with initialization parameters. Passed collections are not * referenced directly, but copied. */

	See In Froyo, dalvik\libcore\security\src\main\java\org\apache\harmony\security In Gingerbread, libcore\luni\src\main\java\org\apache\harmony\security. *See generally, e.g.*: • dalvik\vm\native\InternalNative.c • dalvik\vm\native\java_security_AccessController.c • dalvik\vm\native\dalvik_system_DexFile.c • dalvik\vm\native\java_lang_VMClassLoader.c • For Froyo: • source code files in dalvik\libcore\security\src\main\java\java\security • source code files in dalvik\libcore\security-kernel\src\main\java\java\security • dalvik\libcore\security\src\main\java\org\apache\harmony\security • For Gingerbread • source code files in libcore\luni\src\main\java\java\security • libcore\luni\src\main\java\org\apache\harmony\security

[1-b] in response to detecting the request, determining whether said action is authorized based on permissions associated with a plurality of routines in a calling hierarchy associated with said principal,	In response to detecting the request, Android determines whether the action is authorized based on permissions associated with a plurality of routines in a calling hierarchy associated with the principal.
	
	See, e.g., PolicyEntry.java:
	In Froyo:
	`    /**`
	`     * Checks if passed CodeSource matches this PolicyEntry. Null CodeSource of`
	`     * PolicyEntry implies any CodeSource; non-null CodeSource forwards to its`
	`     * imply() method.`
	`     */`
	`    public boolean impliesCodeSource(CodeSource codeSource) {`
	`        return (cs == null) ? true : cs.implies(codeSource);`
	`    }`
	
	`    /**`
	`     * Checks if specified Principals match this PolicyEntry. Null or empty set`

```
     * of Principals of PolicyEntry implies any Principals; otherwise specified
     * array must contain all Principals of this PolicyEntry.
     */
    public boolean impliesPrincipals(Principal[] prs) {
        return PolicyUtils.matchSubset(principals, prs);
    }

    /**
     * Returns unmodifiable collection of permissions defined by this
     * PolicyEntry, may be <code>null</code>.
     */
    public Collection<Permission> getPermissions() {
        return permissions;
        }
```

In Gingerbread:

```
    /**
     * Checks if passed CodeSource matches this PolicyEntry. Null CodeSource of
     * PolicyEntry implies any CodeSource; non-null CodeSource forwards to its
     * imply() method.
     */
    public boolean impliesCodeSource(CodeSource codeSource) {
        if (cs == null) {
            return true;
        }

        if (codeSource == null) {
            return false;
        }
        return cs.implies(normalizeCodeSource(codeSource));
    }

    private CodeSource normalizeCodeSource(CodeSource codeSource) {
```

	```
	    URL codeSourceURL = PolicyUtils.normalizeURL(codeSource.getLocation());
	    CodeSource result = codeSource;

	    if (codeSourceURL != codeSource.getLocation()) {
	        // URL was normalized - recreate codeSource with new URL
	        CodeSigner[] signers = codeSource.getCodeSigners();
	        if (signers == null) {
	            result = new CodeSource(codeSourceURL, codeSource
	                    .getCertificates());
	        } else {
	            result = new CodeSource(codeSourceURL, signers);
	        }
	    }
	    return result;
	}

	/**
	 * Checks if specified Principals match this PolicyEntry. Null or empty set
	 * of Principals of PolicyEntry implies any Principals; otherwise specified
	 * array must contain all Principals of this PolicyEntry.
	 */
	public boolean impliesPrincipals(Principal[] prs) {
	    return PolicyUtils.matchSubset(principals, prs);
	}

	/**
	 * Returns unmodifiable collection of permissions defined by this
	 * PolicyEntry, may be <code>null</code>.
	 */
	public Collection<Permission> getPermissions() {
	    return permissions;
	}
	```

	See In Froyo, dalvik\libcore\security\src\main\java\org\apache\harmony\security In Gingerbread, libcore\luni\src\main\java\org\apache\harmony\security. Regarding the "calling hierarchy associated with said principal," *see, e.g.*: http://developer.android.com/reference/java/security/AccessController.html static AccessControlContext getContext() Returns the `AccessControlContext` for the current `Thread` including the inherited access control context of the thread that spawned the current thread (recursively). Android Developer Tools available at http://developer.android.com/reference/java/security/AccessControlContext.html *See also* , *e.g.*, java.security.AccessController: /* * java.security.AccessController */ #include "Dalvik.h" #include "native/InternalNativePriv.h" /* * private static ProtectionDomain[] getStackDomains() * * Return an array of ProtectionDomain objects from the classes of the * methods on the stack. Ignore reflection frames. Stop at the first * privileged frame we see. */ static void Dalvik_java_security_AccessController_getStackDomains(const u4* args, JValue* pResult) { UNUSED_PARAMETER(args);

```
const Method** methods = NULL;
int length;

/*
 * Get an array with the stack trace in it.
 */
if (!dvmCreateStackTraceArray(dvmThreadSelf()->curFrame, &methods, &length))
{
    LOGE("Failed to create stack trace array\n");
    dvmThrowException("Ljava/lang/InternalError;", NULL);
    RETURN_VOID();
}

//int i;
//LOGI("dvmCreateStackTraceArray results:\n");
//for (i = 0; i < length; i++)
//    LOGI(" %2d: %s.%s\n", i, methods[i]->clazz->name, methods[i]->name);

/*
 * Generate a list of ProtectionDomain objects from the frames that
 * we're interested in.  Skip the first two methods (this method, and
 * the one that called us), and ignore reflection frames.  Stop on the
 * frame *after* the first privileged frame we see as we walk up.
 *
 * We create a new array, probably over-allocated, and fill in the
 * stuff we want.  We could also just run the list twice, but the
 * costs of the per-frame tests could be more expensive than the
 * second alloc.  (We could also allocate it on the stack using C99
 * array creation, but it's not guaranteed to fit.)
 *
 * The array we return doesn't include null ProtectionDomain objects,
 * so we skip those here.
```

	```
	 */
	Object** subSet = (Object**) malloc((length-2) * sizeof(Object*));
	if (subSet == NULL) {
	    LOGE("Failed to allocate subSet (length=%d)\n", length);
	    free(methods);
	    dvmThrowException("Ljava/lang/InternalError;", NULL);
	    RETURN_VOID();
	}
	int idx, subIdx = 0;
	for (idx = 2; idx < length; idx++) {
	    const Method* meth = methods[idx];
	    Object* pd;

	    if (dvmIsReflectionMethod(meth))
	        continue;

	    if (dvmIsPrivilegedMethod(meth)) {
	        /* find nearest non-reflection frame; note we skip priv frame */
	        //LOGI("GSD priv frame at %s.%s\n", meth->clazz->name, meth->name);
	        while (++idx < length && dvmIsReflectionMethod(methods[idx]))
	            ;
	        length = idx;       // stomp length to end loop
	        meth = methods[idx];
	    }

	    /* get the pd object from the method's class */
	    assert(gDvm.offJavaLangClass_pd != 0);
	    pd = dvmGetFieldObject((Object*) meth->clazz,
	            gDvm.offJavaLangClass_pd);
	    //LOGI("FOUND '%s' pd=%p\n", meth->clazz->name, pd);
	    if (pd != NULL)
	        subSet[subIdx++] = pd;
	}
	```

```
    //LOGI("subSet:\n");
    //for (i = 0; i < subIdx; i++)
    //    LOGI("  %2d: %s\n", i, subSet[i]->clazz->name);

    /*
     * Create an array object to contain "subSet".
     */
    ClassObject* pdArrayClass = NULL;
    ArrayObject* domains = NULL;
    pdArrayClass = dvmFindArrayClass("[Ljava/security/ProtectionDomain;", NULL);
    if (pdArrayClass == NULL) {
        LOGW("Unable to find ProtectionDomain class for array\n");
        goto bail;
    }
    domains = dvmAllocArray(pdArrayClass, subIdx, kObjectArrayRefWidth,
                ALLOC_DEFAULT);
    if (domains == NULL) {
        LOGW("Unable to allocate pd array (%d elems)\n", subIdx);
        goto bail;
    }
```

In Froyo:

```
    /* copy the ProtectionDomain objects out */
    Object** objects = (Object**) domains->contents;
    for (idx = 0; idx < subIdx; idx++)
        *objects++ = subSet[idx];
```

In Gingerbread:

```
    /* copy the ProtectionDomain objects out */
    memcpy(domains->contents, subSet, subIdx * sizeof(Object *));
    dvmWriteBarrierArray(domains, 0, subIdx);
```

	In both Froyo and Gingerbread:
	`bail:`
	`    free(subSet);`
	`    free(methods);`
	`    dvmReleaseTrackedAlloc((Object*) domains, NULL);`
	`    RETURN_PTR(domains);`
	`}`
	
	`const DalvikNativeMethod dvm_java_security_AccessController[] = {`
	`    { "getStackDomains",    "()[Ljava/security/ProtectionDomain;",`
	`        Dalvik_java_security_AccessController_getStackDomains },`
	`    { NULL, NULL, NULL },`
	`};`
	
	dalvik\vm\native\java_security_AccessController.c.
	
	See also, e.g.:
	dalvik\vm\native\dalvik_system_DexFile.c:
	`/*`
	` * private static Class defineClass(String name, ClassLoader loader,`
	` *     int cookie, ProtectionDomain pd)`
	` *`
	` * Load a class from a DEX file.  This is roughly equivalent to defineClass()`
	` * in a regular VM -- it's invoked by the class loader to cause the`
	` * creation of a specific class.  The difference is that the search for and`
	` * reading of the bytes is done within the VM.`
	` *`
	` * The class name is a "binary name", e.g. "java.lang.String".`
	` *`
	` * Returns a null pointer with no exception if the class was not found.`
	` * Throws an exception on other failures.`
	` */`
	`static void Dalvik_dalvik_system_DexFile_defineClass(const u4* args,`

	```
	    JValue* pResult)
	{
	    StringObject* nameObj = (StringObject*) args[0];
	    Object* loader = (Object*) args[1];
	    int cookie = args[2];
	    Object* pd = (Object*) args[3];
	    ClassObject* clazz = NULL;
	    DexOrJar* pDexOrJar = (DexOrJar*) cookie;
	    DvmDex* pDvmDex;
	    char* name;
	    char* descriptor;

	    name = dvmCreateCstrFromString(nameObj);
	    descriptor = dvmDotToDescriptor(name);
	    LOGV("--- Explicit class load '%s' 0x%08x\n", descriptor, cookie);
	    free(name);

	    if (!validateCookie(cookie))
	        RETURN_VOID();

	    if (pDexOrJar->isDex)
	        pDvmDex = dvmGetRawDexFileDex(pDexOrJar->pRawDexFile);
	    else
	        pDvmDex = dvmGetJarFileDex(pDexOrJar->pJarFile);

	    /* once we load something, we can't unmap the storage */
	    pDexOrJar->okayToFree = false;

	    clazz = dvmDefineClass(pDvmDex, descriptor, loader);
	    Thread* self = dvmThreadSelf();
	    if (dvmCheckException(self)) {
	        /*
	         * If we threw a "class not found" exception, stifle it, since the
	```

	``
* contract in the higher method says we simply return null if
* the class is not found.
*/
Object* excep = dvmGetException(self);
if (strcmp(excep->clazz->descriptor,
"Ljava/lang/ClassNotFoundException;") == 0 ||
            strcmp(excep->clazz->descriptor,
                "Ljava/lang/NoClassDefFoundError;") == 0)
        {
            dvmClearException(self);
        }
        clazz = NULL;
    }

    /*
     * Set the ProtectionDomain -- do we need this to happen before we
     * link the class and make it available? If so, we need to pass it
     * through dvmDefineClass (and figure out some other
     * stuff, like where it comes from for bootstrap classes).
     */
    if (clazz != NULL) {
        //LOGI("SETTING pd '%s' to %p\n", clazz->descriptor, pd);
        dvmSetFieldObject((Object*) clazz, gDvm.offJavaLangClass_pd, pd);
    }

    free(descriptor);
    RETURN_PTR(clazz);
}
``` dalvik\vm\native\java_lang_VMClassLoader.c: ```
/*
 * java.lang.VMClassLoader
 */
``` |
```

	```
	…
	/*
	 * static Class defineClass(ClassLoader cl, String name,
	 *     byte[] data, int offset, int len, ProtectionDomain pd)
	 *     throws ClassFormatError
	 *
	 * Convert an array of bytes to a Class object.
	 */
	static void Dalvik_java_lang_VMClassLoader_defineClass(const u4* args,
	    JValue* pResult)
	{
	    Object* loader = (Object*) args[0];
	    StringObject* nameObj = (StringObject*) args[1];
	    const u1* data = (const u1*) args[2];
	    int offset = args[3];
	    int len = args[4];
	    Object* pd = (Object*) args[5];
	    char* name = NULL;

	    name = dvmCreateCstrFromString(nameObj);
	    LOGE("ERROR: defineClass(%p, %s, %p, %d, %d, %p)\n",
	        loader, name, data, offset, len, pd);
	    dvmThrowException("Ljava/lang/UnsupportedOperationException;",
	        "can't load this type of class file");

	    free(name);
	    RETURN_VOID();
	}

	/*
	 * static Class defineClass(ClassLoader cl, byte[] data, int offset,
	 *     int len, ProtectionDomain pd)
	 *     throws ClassFormatError
	```

```
 *
 * Convert an array of bytes to a Class object. Deprecated version of
 * previous method, lacks name parameter.
 */
static void Dalvik_java_lang_VMClassLoader_defineClass2(const u4* args,
    JValue* pResult)
{
    Object* loader = (Object*) args[0];
    const u1* data = (const u1*) args[1];
    int offset = args[2];
    int len = args[3];
    Object* pd = (Object*) args[4];

    LOGE("ERROR: defineClass(%p, %p, %d, %d, %p)\n",
        loader, data, offset, len, pd);
    dvmThrowException("Ljava/lang/UnsupportedOperationException;",
        "can't load this type of class file");

    RETURN_VOID();
}
```

In Froyo, dalvik\libcore\luni\src\main\java\java\lang\SecurityManager.java
In Gingerbread, libcore\luni\src\main\java\java\lang\SecurityManager.java:

```
/**
 * <strong>Warning:</strong> security managers do <strong>not</strong> provide a
 * secure environment for executing untrusted code. Untrusted code cannot be
 * safely isolated within the Dalvik VM.
 *
 * <p>Provides security verification facilities for applications. {@code
 * SecurityManager} contains a set of {@code checkXXX} methods which determine
 * if it is safe to perform a specific operation such as establishing network
 * connections, modifying files, and many more. In general, these methods simply
 * return if they allow the application to perform the operation; if an
 * operation is not allowed, then they throw a {@link SecurityException}. The
 * only exception is {@link #checkTopLevelWindow(Object)}, which returns a
 * boolean to indicate permission.
 */
public class SecurityManager {
…
    /**
     * Checks whether the calling thread is allowed to access the resource being
     * guarded by the specified permission object.
     *
     * @param permission
     *            the permission to check.
     * @throws SecurityException
     *             if the requested {@code permission} is denied according to
     *             the current security policy.
     */
    public void checkPermission(Permission permission) {
        try {
            inCheck = true;
            AccessController.checkPermission(permission);
        } finally {
            inCheck = false;
        }
    }

    /**
     * Checks whether the specified security context is allowed to access the
     * resource being guarded by the specified permission object.
     *
     * @param permission
     *            the permission to check.
     * @param context
     *            the security context for which to check permission.
```

	```
	 * @throws SecurityException
	 *           if {@code context} is not an instance of {@code
	 *           AccessControlContext} or if the requested {@code permission}
	 *           is denied for {@code context} according to the current
	 *           security policy.
	 */
	public void checkPermission(Permission permission, Object context) {
	   try {
	      inCheck = true;
	      // Must be an AccessControlContext. If we don't check
	      // this, then applications could pass in an arbitrary
	      // object which circumvents the security check.
	      if (context instanceof AccessControlContext) {
	         ((AccessControlContext) context).checkPermission(permission);
	      } else {
	         throw new SecurityException();
	      }
	   } finally {
	      inCheck = false;
	   }
	}
	}
	```
	In Froyo, dalvik\libcore\security-kernel\src\main\java\java\security\AccessController.java
	In Gingerbread, libcore\luni\src\main\java\java\security\AccessController.java:
	```
	/**
	 * Checks the specified permission against the vm's current security policy.
	 * The check is performed in the context of the current thread. This method
	 * returns silently if the permission is granted, otherwise an {@code
	 * AccessControlException} is thrown.
	 * <p>
	 * A permission is considered granted if every {@link ProtectionDomain} in
	 * the current execution context has been granted the specified permission.
	```

	```
	     * If privileged operations are on the execution context, only the {@code
	     * ProtectionDomain}s from the last privileged operation are taken into
	     * account.
	     * <p>
	     * This method delegates the permission check to
	     * {@link AccessControlContext#checkPermission(Permission)} on the current
	     * callers' context obtained by {@link #getContext()}.
	     *
	     * @param permission
	     *            the permission to check against the policy
	     * @throws AccessControlException
	     *             if the specified permission is not granted
	     * @throws NullPointerException
	     *             if the specified permission is {@code null}
	     * @see AccessControlContext#checkPermission(Permission)
	     *
	     * @since Android 1.0
	     */
	    public static void checkPermission(Permission permission)
	            throws AccessControlException {
	        if (permission == null) {
	            throw new NullPointerException("permission == null");
	        }

	        getContext().checkPermission(permission);
	    }
	```
	In Froyo, dalvik\libcore\security-kernel\src\main\java\java\security\AccessControlContext.java
	In Gingerbread, libcore\luni\src\main\java\java\security\AccessControlContext.java:
	```
	    // List of ProtectionDomains wrapped by the AccessControlContext
	    // It has the following characteristics:
	    //     - 'context' can not be null
	```

	`//     - never contains null(s)`
	`//     - all elements are unique (no dups)`
	`ProtectionDomain[] context;`
	`…`
	`   /**`
	`    * Checks the specified permission against the vm's current security policy.`
	`    * The check is based on this {@code AccessControlContext} as opposed to the`
	`    * {@link AccessController#checkPermission(Permission)} method which`
	`    * performs access checks based on the context of the current thread. This`
	`    * method returns silently if the permission is granted, otherwise an`
	`    * {@code AccessControlException} is thrown.`
	`    * <p>`
	`    * A permission is considered granted if every {@link ProtectionDomain} in`
	`    * this context has been granted the specified permission.`
	`    * <p>`
	`    * If privileged operations are on the call stack, only the {@code`
	`    * ProtectionDomain}s from the last privileged operation are taken into`
	`    * account.`
	`    * <p>`
	`    * If inherited methods are on the call stack, the protection domains of the`
	`    * declaring classes are checked, not the protection domains of the classes`
	`    * on which the method is invoked.`
	`    *`
	`    * @param perm`
	`    *            the permission to check against the policy`
	`    * @throws AccessControlException`
	`    *            if the specified permission is not granted`
	`    * @throws NullPointerException`
	`    *            if the specified permission is {@code null}`
	`    * @see AccessController#checkPermission(Permission)`
	`    */`
	`   public void checkPermission(Permission perm) throws AccessControlException {`
	`      if (perm == null) {`

	```
	            throw new NullPointerException("Permission cannot be null");
	        }
	        for (int i = 0; i < context.length; i++) {
	            if (!context[i].implies(perm)) {
	                throw new AccessControlException("Permission check failed "
	                        + perm, perm);
	            }
	        }
	        if (inherited != null) {
	            inherited.checkPermission(perm);
	        }
	    }
	```
	See also, e.g:
	• dalvik\tests\025-access-controller\expected.txt;
	• dalvik\tests\025-access-controller\src\Main.java;
	• dalvik\tests\025-access-controller\src\Privvy.java;
	• In Froyo
	  o dalvik\libcore-disabled\sound\src\main\java\org\apache\harmony\sound\utils\ProviderService.java;
	  o dalvik\libcore\security\src\test\java\tests\security
	• In Gingerbread
	  o libcore\luni\src\test\java\tests\security.

[1-c] wherein said permissions are associated with said plurality of routines based on a first association between protection domains and permissions.	The permissions are associated with the routines based on an association between protection domains and permissions. *See* Claim 1-b, *supra*. *See* In Froyo, dalvik\libcore\security\src\main\java\java\security\ProtectionDomain.java In Gingerbread, libcore\luni\src\main\java\java\security\ProtectionDomain.java:

	* {@code ProtectionDomain} represents all permissions that are granted to a
	 * specific code source. The {@link ClassLoader} associates each class with the
	 * corresponding {@code ProtectionDomain}, depending on the location and the
	 * certificates (encapsulates in {@link CodeSource}) it loads the code from.
	 * <p>
	 * A class belongs to exactly one protection domain and the protection domain
	 * can not be changed during the lifetime of the class.
	 */
	public class ProtectionDomain {
	
	    // CodeSource for this ProtectionDomain
	    private CodeSource codeSource;
	
	    // Static permissions for this ProtectionDomain
	    private PermissionCollection permissions;
	
	    // ClassLoader
	    private ClassLoader classLoader;
	
	    // Set of principals associated with this ProtectionDomain
	    private Principal[] principals;
	
	    // false if this ProtectionDomain was constructed with static
	    // permissions, true otherwise.
	    private boolean dynamicPerms;

The '476 Patent	**Infringed By**
2. The method of claim 1, wherein:	*See* Claim 1, *supra*.

The ’476 Patent	Infringed By
10. A computer-readable medium carrying one or more sequences of one or more instructions, the one or more sequences of the one or more instructions including instructions which, when executed by one or more processors, causes the one or more processors to perform the steps of:	The Accused Instrumentalities include devices that store, distribute, or run Android or the Android SDK, including websites, servers, and mobile devices. These encompass a computer readable medium carrying one or more sequences of one or more instructions. See corresponding elements of claim 1, *supra*.
detecting when a request for an action is made by a principal; and	See corresponding elements of claim 1, *supra*.
in response to detecting the request, determining whether said action is authorized based on permissions associated with a plurality of routines in a calling hierarchy associated with said principal, wherein said permissions are associated with said plurality of routines based on a first association between protection domains and permissions.	See corresponding elements of claims 1 and 5, *supra*.

The ’476 Patent	Infringed By
11. The computer-readable medium of claim 10, wherein:	*See* corresponding elements of claims 1 and 10, *supra.*
the step of detecting when a request for an action is made includes detecting when a request for an action is made by a thread; and	*See* corresponding elements of claims 1, 2, and 10, *supra.*
the step of determining whether said	*See* corresponding elements of claims 1, 2, and 10, *supra.*

The '476 Patent	Infringed By
action is authorized includes determining whether said action is authorized based on an association between permissions and a plurality of routines in a calling hierarchy associated with said thread.	

The '476 Patent	Infringed By
12. The computer readable medium of claim 10, wherein:	*See* corresponding elements of claims 1 and 10, *supra*.
the calling hierarchy includes a first routine; and	*See* corresponding elements of claims 1, 3, and 10, *supra*.
the step of determining whether said action is authorized further includes determining whether a permission required to perform said action is encompassed by at least one permission associated with said first routine.	*See* corresponding elements of claims 1, 3, and 10, *supra*.

The '476 Patent	Infringed By
13. The computer readable medium of claim 10, wherein the step of determining whether said action is authorized further includes determining whether a permission required to perform said action is encompassed by at least one permission associated with each routine in said calling hierarchy.	*See* corresponding elements of claims 1 and 10, *supra*.

The ’476 Patent	Infringed By
14. A computer-readable medium bearing instructions for providing security, the instructions including instructions for performing the steps of:	*See* corresponding element of claim 1, *supra*.
detecting when a request for an action is made by a principal;	*See* corresponding element of claim 1, *supra*.
determining whether said action is authorized based on an association between permissions and a plurality of routines in a calling hierarchy associated with said principal;	*See* corresponding element of claim 1, *supra*.
wherein each routine of said plurality of routines is associated with a class; and	*See* corresponding element of claim 5, *supra*.
wherein said association between permissions and said plurality of routines is based on a second association between classes and protection domains.	*See* corresponding element of claim 5, *supra*.

The ’476 Patent	Infringed By
15. A computer-readable medium carrying one or more sequences of one or more instructions, the one or more sequences of the one or more instructions including instructions which, when executed by one or more processors, causes the one or more processors to perform the steps	*See* corresponding element of claim 1, *supra*.

The '476 Patent	Infringed By
of:	
detecting when a request for an action is made by a principal; and	*See* corresponding element of claim 1, *supra*.
in response to detecting the request, determining whether said action is authorized based on permissions associated with a plurality of routines in a calling hierarchy associated with said principal, wherein a first routine in said calling hierarchy is privileged; and	*See* corresponding elements of claims 1 and 6, *supra*.
wherein the step of determining whether said action is authorized further includes determining whether a permission required to perform said action is encompassed by at least one permission associated with each routine in said calling hierarchy between and including said first routine and a second routine in said calling hierarchy, wherein said second routine is invoked after said first routine, wherein said second routine is a routine for performing said requested action.	*See* corresponding elements of claims 1 and 6, *supra*.

The '476 Patent	Infringed By
16. The computer readable medium of claim 15, wherein the step of determining whether said permission	*See* claims 7 and 15, *supra*.